Model Representation

$x^{(i)}$ – ith input variable or feature (in this case, living area)
$y^{(i)}$ – ith output or target variable that we are trying to predict (price)
$(x^{(i)}, y^{(i)})$ – One training example.

Thus, the dataset we use to learn a function is called a training set because it is basically a list of m training examples $(x^{(i)}, y^{(i)})\ ; i = 1, \dots, m$

Our goal is to use the training set and learn a function h(x) (hypothesis function) that is a good predictor for the corresponding values of y.

If the target variable is continuous, then this learning problem is a regression problem. If discrete, it is a classification problem.

In a simple linear regression, the hypothesis function can be represented by $h_\theta(x) = \theta_0 + \theta_1 x_i$

Cost Function

This function allows us to simply measure the accuracy of our hypothesis function. It takes a fancier average of the squared difference between the predicted values and actual values.

We can represent the cost function as

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

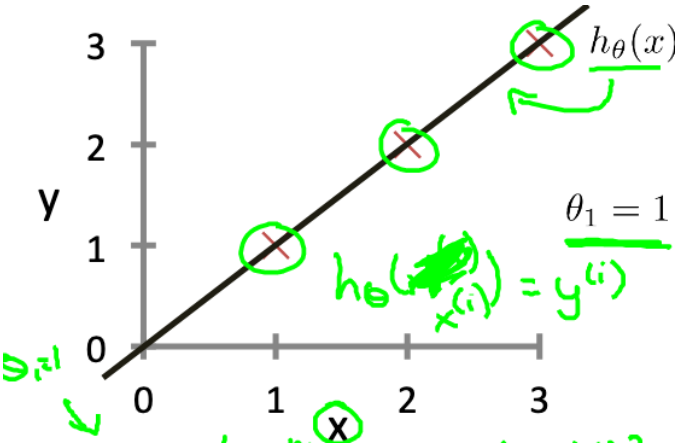
This function is otherwise called the "Squared error function", or "Mean squared error". The goal is to minimize $J(\theta_0, \theta_1)$ with respect to θ_0, θ_1.

Cost Function Intuition – I

Let's assume that $\theta_0 = 0$. In this case, $h_\theta(x) = \theta_1 x$, and the cost function would be

$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_1 x^{(i)} - y_i)^2$$

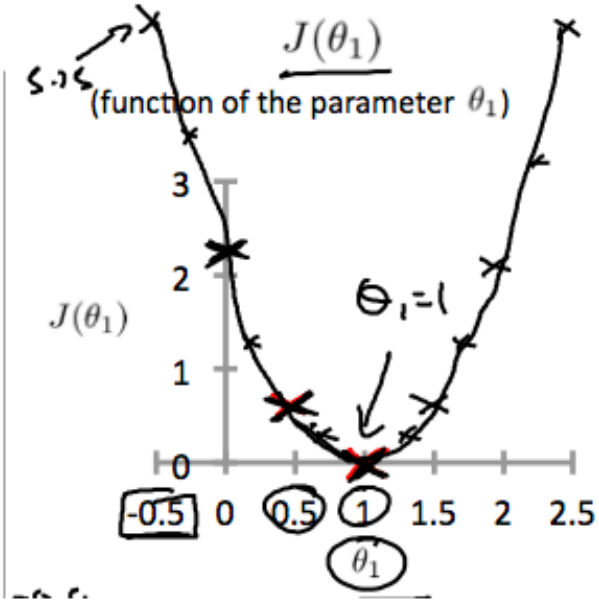
Thus, we want to minimize $J(\theta_1)$ with respect to θ_1.



The graph illustrates that the hypothesis function predicts all the input variables correctly. Thus, the output of the cost function is

$$\frac{1}{2m}[\,(1-1)^2 + (2-2)^2 + (3-3)^2\,]$$
$$\therefore\ J(\theta_1) = 0$$

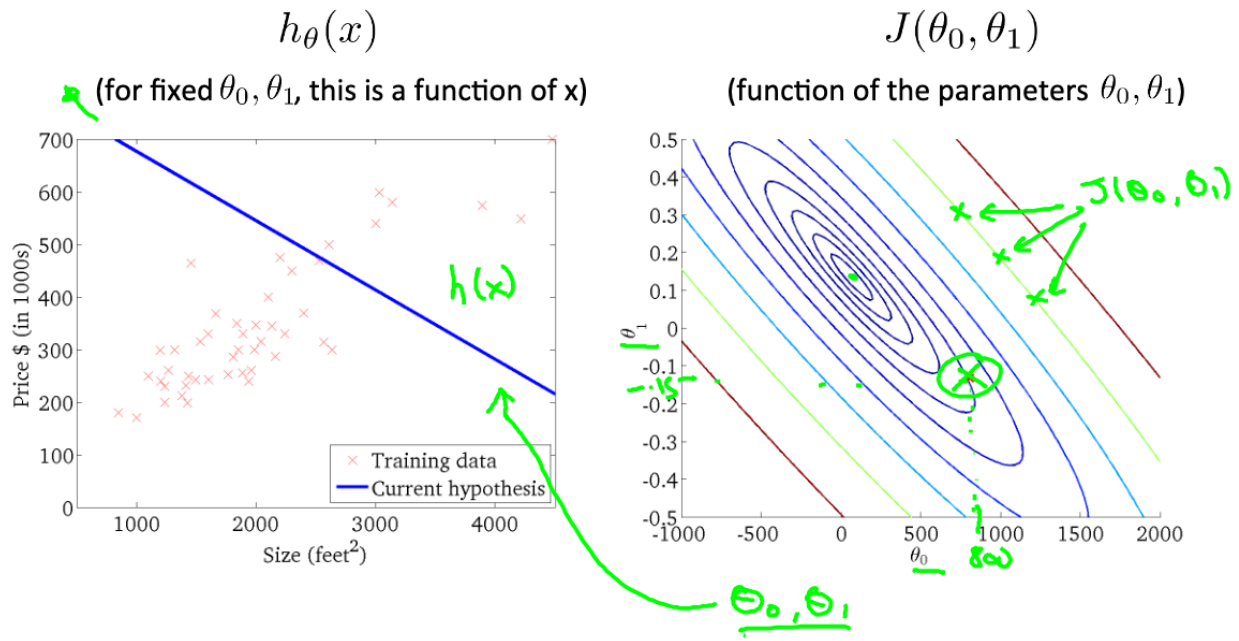
Essentially, we choose a slope for the hypothesis function and calculate the cost function error of that slope.
A different slope yields a different cost as the distances between each prediction and actual value will change. Repeating this process for many different slopes will result in several ordered pairs of the slope and the cost function error that we can plot as shown below. This parabola indicates that $\theta_1 = 1$ is the best choice.



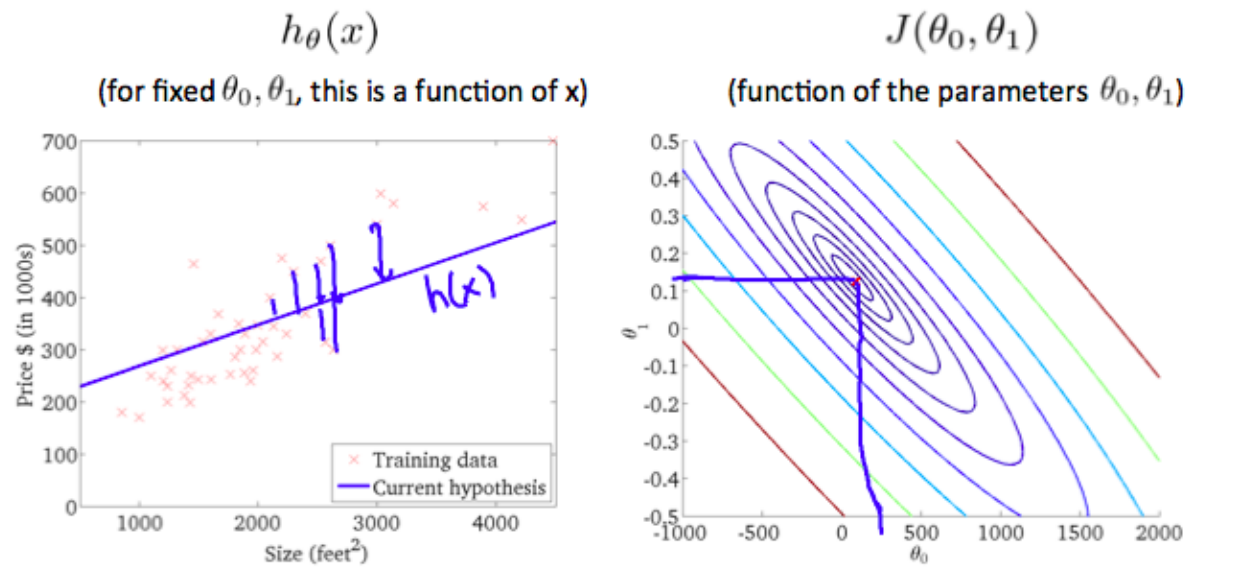
Cost Function Intuition – II

In many cases, the y-intercept of the hypothesis function will not be 0. Thus, we will have to minimize $J(\theta_0, \theta_1)$ with respect to θ_0 *and* θ_1. However, the process from Intuition I still applies: Use the training set to learn a hypothesis function, calculate the cost function error using the cost function, repeat the process using several different $\theta_0's$ *and* $\theta_1's$, and choose the pair that minimizes the cost function.

We can use a contour figure to visualize the cost function errors for different $\theta_0's$ *and* $\theta_1's$.



The three green points found on the green line above have the same value for $J(\theta_0, \theta_1)$. Reducing this error gets us closer to the center of the contour plot.



Parameter Learning

Gradient Descent

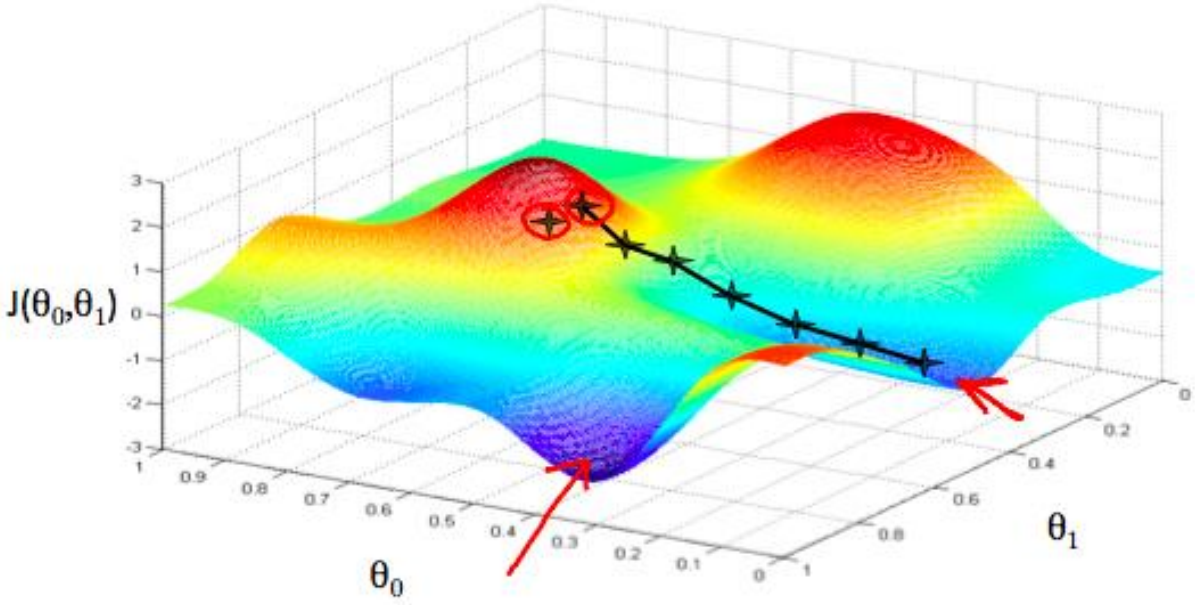
This is a more general algorithm that minimizes the cost function J and is used in all areas of Machine Learning.

Let's say that we have a cost function, not necessarily a linear regression cost function, to minimize $J(\theta_0, \theta_1)$ with respect to θ_0 and θ_1. Note that Gradient Descent also applies to a cost function with more than two parameters such as $J(\theta_0, \theta_1, \dots, \theta_n)$. For brevity's sake, we will assume that we only have two parameters.

Here is the basic outline:

1. Start with some θ_0 and θ_1
2. Keep changing θ_0 and θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

Let's visualize this process by graphing several $\theta_0's$ and $\theta_1's$, and the cost resulting from a particular set of parameters.



We know we have done a very good job of estimating the parameters in the hypothesis function when our cost function is at the very bottom of the pits in our graph, the global minimum. The way to do this is by taking the derivative of the cost function because it will give us the direction to, we need to move towards.

The algorithm is:

Repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (for\ j = 0\ and\ j = 1)$$

α is a learning parameter that determines the size of each step. A smaller α results in a smaller step, and larger α results in a larger step. The partial derivative determines the direction of each step.

Two important things to note.

1. Depending on the parameters the algorithm starts with, you could end up at a different minimum.
2. The correct implementation is to update the parameters simultaneously. Thus, calculate the new values of all the parameters first, and then update the parameters accordingly.

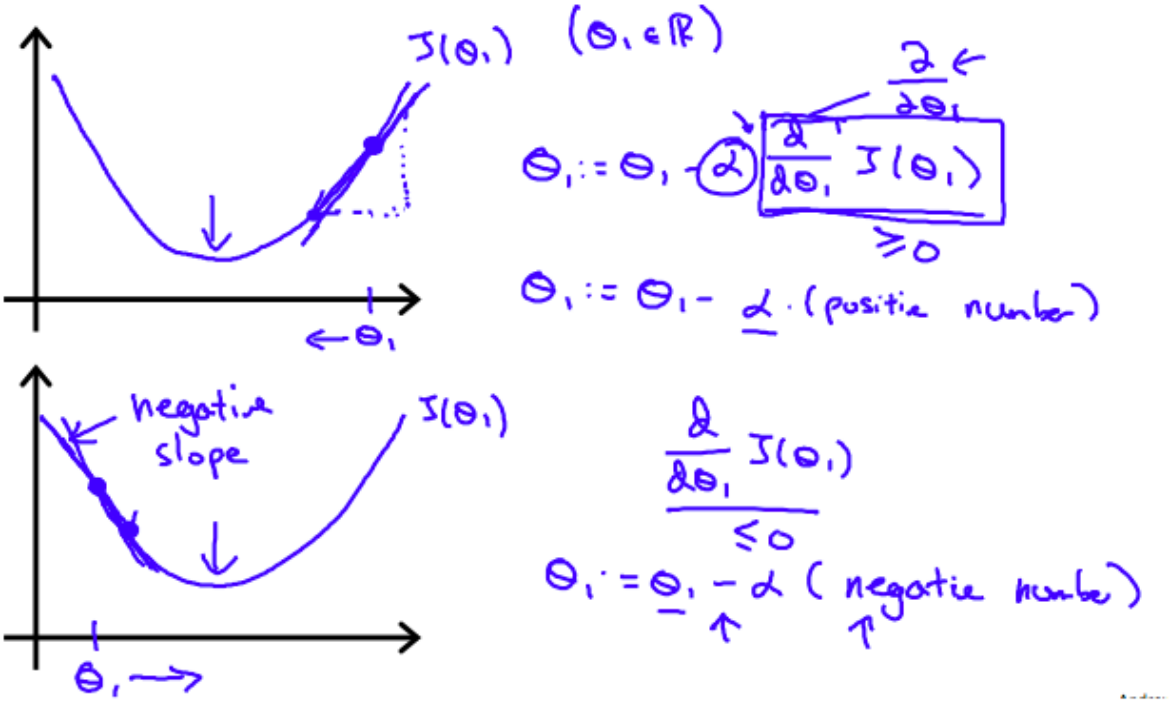
Gradient Descent Intuition

Let's develop a deeper understanding of the formula using a very simple cost function that only has one parameter. Thus, our formula for a single parameter is

Repeat until convergence:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

The derivative helps us understand whether or not the function at a particular θ_1 is increasing or decreasing. If the cost function is a parabola as shown below, and the derivative at θ_1 is positive, then the new θ_1 should move to the left. Thus, θ_1 should decrease in order to get closer to the cost function's minimum.

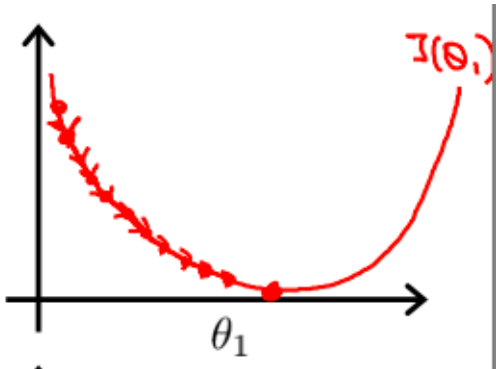


Similarly, if the derivative at θ_1 is negative, and given our cost function in the above picture, θ_1 should increase. In both cases, θ_1 moves in the direction that gets it closer to the minimum of the cost function. The direction θ_1 will move depends on the sign of the derivative, and because the θ_1 moves in the opposite direction, the product of alpha and the partial derivative are **subtracted** from the initial θ_1.

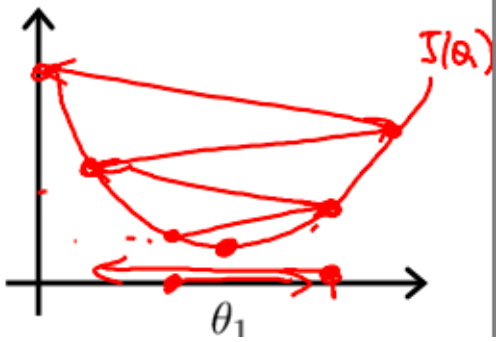
Understanding α

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



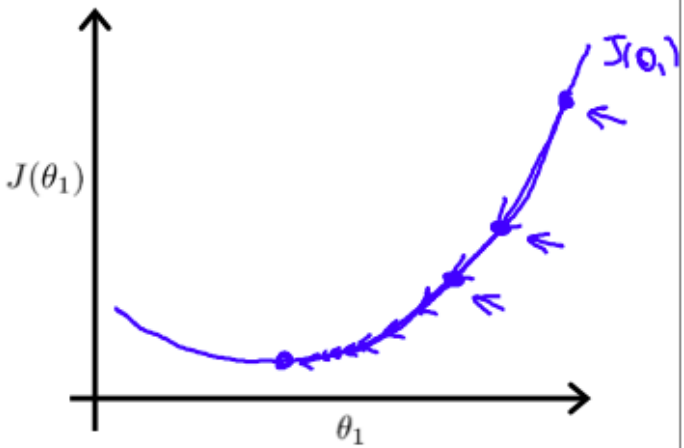
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time



There are a few important things to note. These things do not apply to parabola drawn above but to functions that have multiple extremums and even saddle points.

1. Just because θ_1, and essentially, the derivative of θ_1 is approaching zero, it does not mean that the parameter is approaching the cost function's minimum. It is possible that θ_1 is approaching either a maximum or a saddle point (not an extremum), points where the derivative of θ_1 will equal zero.
2. If the θ_1 converges and settles at a maximum point, then θ_1 is the opposite of what we wanted to do, find the minimum.
3. If θ_1 converges towards a saddle point, then we failed to find a minimum. Because the derivative at the saddle point is zero, the new value of θ_1 will equal the previous value and that is where θ_1 will settle.

Gradient Descent for Linear Regression

When applied to the case of linear regression, we can write a new form of the gradient descent equation.

We have the Gradient Descent algorithm for linear regression with only two parameters,

Repeat Until Convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \ (for\ j = 0\ and\ j = 1),$$

And the Linear regression model along with its cost function

$$h_\theta(x) = \theta_0 + \theta_1 x_i$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Substituting the $J(\theta_0, \theta_1)$ in the gradient descent algorithm with the linear regression model and cost function, the partial derivative term becomes

$$\frac{\partial}{\partial \theta_j} \left[\frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_i - y_i)^2 \right]$$

Thus, the gradient descent algorithm can be re-written as:
Repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha * \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha * \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x_i) - y_i) * x_i$$

}

While the gradient descent can be susceptible to local minima in general, linear regression has only one global, and no other local, optima. Therefore, the algorithm always converges (assuming the learning rate α is not too large) to the global minimum as shown below

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)